A FILE MANAGEMENT SYSTEM FOR RELATIONAL DATA BASE

By

NALINI KANTA RATHA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

JUNE, 1984

A FILE MANAGEMENT SYSTEM FOR RELATIONAL DATA BASE

A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of

MASTER OF TECHNOLOGY

By

NALINI KANTA RATHA

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

JUNE, 1984

to

my parents

-Nalini

C E R T I F I C A T E

This is to certify that the thesis entitled "A File Management System for Relational Data Base" is a report of work carried out under my supervision by Nalini Kanta Ratha, and that it has not been submitted elsewhere for a degree.

Rajeev Sangal

(Dr. Rajeev Sangal)

Assistant Professor

Department of Computer Science

and Engineering

INDIAN INSTITUTE OF TECHNOLOGY

KANPUR-208016

KANPUR,

JUNE,1984

ABSTRACT

A file management system has been implemented on top of operating system CP/M 1.4 for microcomputers. It is a part of a group project titled "Implementation of RDB/M - a relational data base on microcomputers ". Two types of file structures namely (i) Hashed, and (ii) ISAM have been provided. They have been designed particularly for floppy disk as the storage medium. However, they can be adapted to other media by simply changing the module that decides on physical placement of blocks. A set of routines have been written to access both the file structure by means of a uniform interface. An efficient buffer management scheme has been adopted to minimize secondary storage accesses. Performance analysis of both the file organizations for the particular implemented structure has been carried out.

A C K N O W L E D G E M E N T

I take this oppurtunity to express my deep sense of gratitude and heart-felt thanks to Dr. Rajeev Sangal for his inspiring guidance and suggestions at every stage of the thesis.

I am thankful to all those who have made my stay at IIT/K, pleasant and memorable.

Special mention must be made of the following :

- all the members of the DB group (all of us would surely miss the weekly meetings)
- all the classmates
- Mr. S.K.DAS having helped to keep the Cromemco system up
- Mr. J.Senthil who has been a constant source of inspiration
- all Oriya friends of mine

JUNE, 1984 (N.K.Ratha)

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 MOTIVATION

An efficient implementation of a data base management system is dependent on the underlying secondary storage structure. Since the data bases cannot generally be accomodated in the main memory, time and again the secondary storage devices will have to be accessed to make retrievals and updates to the data base.

A system that minimizes the secondary accesses would have better performance.

For a microcomputer system the problem is not only with slow secondary storage devices but also very limited main memory and secondary storage space. The commonly used secondary storage device for a microcomputer system is a floppy diskette which has a limited storage capacity and access time is also very high.

In answering a query in a relational data base, a large number of secondary accesses are to be made because of the power of the query languages being as powerful as relational algebra.

Keeping in view, the problems with a microcomputer system the following objectives have been set in designing a storage structure :

(i) Minimize number of disk accesses

(ii) use as less memory space as possible

(iii) provide an efficient access method independent of storage structure

This is an initial attempt to support a relational data base scheme on microcomputers. This has been taken up as a group project. In listing requisite features for a hypothetical relational data base KIM[KIM79] describes nine features. After giving careful thought to the features and needs of such a system on a microcomputer four features have been chosen namely,

(i) to support an efficient file structure and efficient access path to the data base

(ii) an interface for a high level non-procedural language for query, data manipulation, data definition and data control

(iii) integrity control

(iv) recovery from both soft and hard crashes.

This thesis is aimed at providing the first feature in the above list. The prime motivation for taking up such an activity is

(i) abundance of microcomputers and trend towards

distributed computing

(ii) an intersting exercise in implementing a data base to be aware of the problems and issues involved in an implementation.

1.2 RESULTS

The following storage organizations have been implemented :

(1) Hashed

(2) ISAM

An access method interface independent of the underlying structure has been provided. Both the organizations use a buffering scheme to minimize secondary storage accesses.

1.3 DATA BASE MANAGEMENT SYSTEMS

One of the wide spread uses of computers is storing,retrieving data from real life. A data base is a collection of such data stored in a computer. A data base management system(DBMS) is the software which provides an abstract view of the data and allows one or more users to modify the data using operators in the abstract view. It is a generalized tool for manipulating data[FRSB76].

A DBMS is designed with the following objectives :

(1) to provide an abstract integral view of data available to users.

(2) to provide logical data independence and physical data independence

(3) to allow centralized control of data base

(4) to provide a friendly user environment

(5) to ensure quality and integrity of data

(6) to ensure privacy and security

(7) to provide ability to recover the system from failures

The purpose of DBMS is to avoid the trouble of learning the details of data storage and its access mechanism by the user. The user deals with the logical model and does not have to worry about the machine representation of his data. In providing all these, it assures integrity,security and privacy of data. In general a DBMS is an efficient manager of storing,retrieving,updating data in a data base in an integrated manner.

1.4 RELATIONAL DATA MODEL

Data about an entity is the values of attributes that describe it. Once data is captured the question arises how to store it physically in the data base. The model that is adopted has mainly two elements [ULLMA82]

(i) a mathematical notation for expressing data and

relationship, and

(ii) operations on the data that serve to express queries and manipulation of the data

Three standard models that are in use are (i) Relational (ii) Network (iii) Hierarchical.

The mathematical concept behind relational model is the set theoretic relation. This model was first proposed by Codd[CODD70] in 1970. The theoritical basis for this model has been developed by Codd[CODD70]. A relation consists of a set of tuples with each member having the same set of attributes. A relation with simple domain can be represented in a tabular form with no duplicate tuples. The attributes with atomic values are represented by columns in a relation table. A tuple is a mapping from attribute names to value in their domains.

The relational model eliminates the users' needs to know the representation of their data. Hence an efficient storage system could be organized on this model which could be totally transperant to the end users.

A data base system is fully relational if it supports [CODD79]

(1) the structural aspect of the relational model

(2) the insert-update-delete rules

(3) a data sublanguage as powerful as the relational algebra

A typical relation is shown in fig.1.1.

The operations that are required to manipulate data in a relational model are described by Codd in his pioneering paper [CODD70]. Operators to select tuples on a condition, or to connect data across relations are also available besides basic set theoritic operators like union, differrence, cross product.

The collection of relation schemes used to represent information is called relational data base scheme and the time varying collection of tuples belonging to relations in a scheme, all of which can be accessed and updated, is called a relational data base.

A true relational data base should possess the following properties [CHAMB76]

(1) all information is represented by data values. No essential information is contained in invisible connections among records.

(2) at the user interface, no particular access path is preferred over any other.

(3) The user interface is independent of the means by which data is physically stored.

The relational model has many advantages over the other two models :

(1) simple

(2) data inpendence

(3) symmetry i.e. no particular type of query is answered faster than any other type.

(4) Strong theoretical support

The only disadvantage of relational model is that the implementation becomes cumbersome as it tries to take the burden of access to the data away from the user.

There are over a dozen of existing relational systems implemented on main frames and minicomputers. A survey of some of them has been made in [KIM79].

Student Name	Roll No	Course	Sem	Year	Grade
RAM	111111	CS315	I	83-84	A
MOHAN	222222	CS415	II	82-83	B
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.

Fig. 1.1 : A typical relation

1.5 ORGANIZATION OF THE THESIS

Chapter 1 gives an introduction to DBMS and relational model.

Chapter 2 describes physical data organizations with their merits and demerits in general. It briefly describes the operating system for the current implementation and the file structure.

The structure and organization of the two file systems provided and the access methods interface is described in Chapter 3.

In Chapter 4 a functional level description of the file management system is described.

Chapter 5 analyses the need for buffers and the related data structures and describes the buffer allocation algorithm. The performance analysis of the particular file structures is carried out in Chapter 6. The concluding Chapter 7 summarizes work done and throws some light onto future work.

CHAPTER 2

PHYSICAL DATA ORGANIZATION

2.1 FILE SYSTEMS

The simplest way to store a relation of a relational scheme is as a file with records as tuples and fields representing the attributes of the tuples. For a small relation a heap organization would be better in terms of space, but in a micro system heap organization would have a poor response.

A file is a collection of fixed or variable size records. A record is the logical unit of access and for relational model it is a tuple. In a file the collection of attributes that identify a record uniquely is called a key. A primary index on the file consists of collection of entries, each corresponding to a data record containing the value of the key for that record. A secondary index for a set of attributes that may participate in a key for the relation, is a relationship between the domain of these attributes and the set of records in the file.

The basic file organizations are sequential, index sequential(ISAM), direct or hashed, indexed etc..

These organizations are based on primary index. There are few other organizations based on secondary indices such as inverted files, list and multilist structures. A good description of various files and their performance analysis is available in [WEID2].

For a relational data base system the following file organizations are typically provided :
(i) sequential (ii) index sequential (iii) direct (Hashed) (iv) Inverted [KIM79].

For an efficient implementation of the relational system, the storage structure plays an important role because the response time to a query is very much dependent on the underlying storage structure and the access mechanism.

Keeping this in view, two basic structures, namely, (i) Hashed (ii) ISAM have been supported by the current implementation.

2.2 CP/M -The O.S. for current implementation

The file structures depend on the environment in which they play their role. The basic factor in this is the operating system. For the current implementation CP/M (Control Program for Microcomputers) has been chosen as the operating system. The choice was rather obvious because

(i) It is very popular and widely used. Most microcomputers in the country today support this.

(ii) It is very vesatile and can suit any kind of environment. Many standard packages are available on this as utilities.

CP/M is a single user interactive operating system running on Z-80 or INTEL 8080/8085 processor based systems. Details of the operting system are available in [CPMIG],[CPMUG], [CPMAG].

2.3 DISKETTE ORGANIZATION IN CP/M

The widely used secondary device for a microcomuter is a diskette(floppy). The diskettes come in various standard sizes and capacities. The organization on a floppy provide the basic design parameters for the file system. The following are the specifications of relevance to us

> single sided single density,IBM compatible

> 8-inch diameter and soft sectored

> 77 tracks/floppy

> 26 sectors/track of 128 bytes each

> 8 sectors per block

> User available space of 241K bytes

The physically adjacent sectors are not logically sequential sectors in CP/M. This has been done with a view to support systems with lesser memory so that computations on the data from a previous sector are completed before the next sector is available below the head. A cluster allocation map is usually provided to find out the sequential sectors. A sector is the basic unit of transfer between the CPU and the diskette.

2.4 CP/M FILE STRUCTURE

We as designers of a file sysstem would be more intersted in the file structure provided by the operating system. In CP/M version 2.0 and higher both sequential and random files are supported. An ASCII file is a sequence of ASCII characters, each line being marked by a carriage return and line feed. The end of the ASCII file is marked by Control-Z. The file size could be as high as the diskette capacity. A file is divided into logical segments of 16K bytes each called the "EXTENT". Each extent is accessible through a control block called as File Control Block(FCB). The structure of FCB is given in Fig.2.1.

CP/M adopts a named file structure consisting of three parts :

(1) disk drive

(2) One to Eight non blank characters as name

(3) Zero to Three characters as extension

The FCBs are stored in a logically distinct area in the diskette and a directory is maintained by CP/M. Maximum of 64 such entries can be accomodated in a diskette described in the previous section.

DR	F1	F2	..	F8	T1	T2	EX	S1	S2	RC	D0	..	D1	CR	R0	R1	R2

DR => Drive

F1..F8 => File Name

T1..T3 => Extension of file

EX => Extent

R0..R2 => Used for random files

S1,S2 => Reserved for system use

RC => Record Count

D0..D15 => Block nos.

CR => Current Record

Fig. 2.1 : File Control Block

The BDOS(Basic Disk Operating System) module of CP/M provides with various system functions for various file operations such as creating a file, opening a file, deleting

a file, writing and reading a sector of the file etc.. Table 2.1 describes the file handling BDOS functions.

A detailed list of functions and the way to access them is available in [CPM10].

Table 2.1

Function No.	Function Name
(1) 15	Open file
(2) 16	Close file
(3) 17	Search for first
(4) 18	Search for next
(5) 19	Delete file
(6) 20	Read sequential
(7) 21	Write sequential
(8) 22	Make file
(9) 23	Rename file
(10) 26	Set DMA address
(11) 27	Get addr(ALLOC)

CHAPTER 3

FILE ORGANIZATIONS

3.1 TWO FILE STRUCTURES

As mentioned in section 2.1 the two types of file organizations in the current implementation are (i) Hashed (ii) ISAM. In the later sections the structures of both the files are discussed.

3.2 HASHED FILE STRUCTURE

In a hashed file, the records of the file are distributed among a fixed number of buckets. A bucket could be a physical block or more. A bucket directory is maintained where each entry points to the block number for the bucket. The bucket to which a record should go is decided by its key value. The key value is hashed by a hash function which decides uniquely the bucket number.

The important parameters in a hash file design are the following :

(i) Hash function

(ii) Bucket directory size

(iii) Handling bucket overflows

(iv) Obtaining the key value from the attributes

Hash function in the current implementation is the 'MOD' function.

H(k.v.)= (k.v.) MOD (bucket dir size)
where MOD is the usual modulo operator.

Decision on bucket directory has been left to the user. Four different sizes of bucket directory have been provided. Irrespective of the file size the user can chose any one of them. Table given below describes it.

TYPE	Buck Dir Size
(i) SMALL	(i) 17
(ii) MEDIUM	(ii) 37
(iii) LARGE	(iii) 67
(iv) VERY LARGE	(iv) 131

Bucket overflow is handled by chaining. If the current block in the bucket is full a new block is acquired and chained to the current one through the 'NEXTBLK' field. In case of a look up the whole set of blocks linked together are to be searched and the bucket directory points to the header block in the chain.

To get an integer value from the set of key attributes the following procedure has been adopted as described in [KNU3].

(1) Treat the value of attributes as sequence of bits. In case of more than one attribute concatenate the bit sequence in the same order they appear in the relation.

(2) Make groups of 16 bits each

(3) Get the integer value of each member

(4) Sum up the integers obtained in this way.

The stucture of the Hashed file is shown in Fig. 3.1.

The bucket directory has two pointers, one pointing to the header block in the chain, the other points to the free available space in the bucket. The latter is used while writing a record in the bucket, without going through the chain of blocks to find the free space.

The bucket directory and other control information like record size, total number of records in the file, number of records in a physical block, number of attributes in the relation and their sizes in bytes, key fields are stored in the first block of the file called as control block. This information is loaded onto main memory while opening the file.

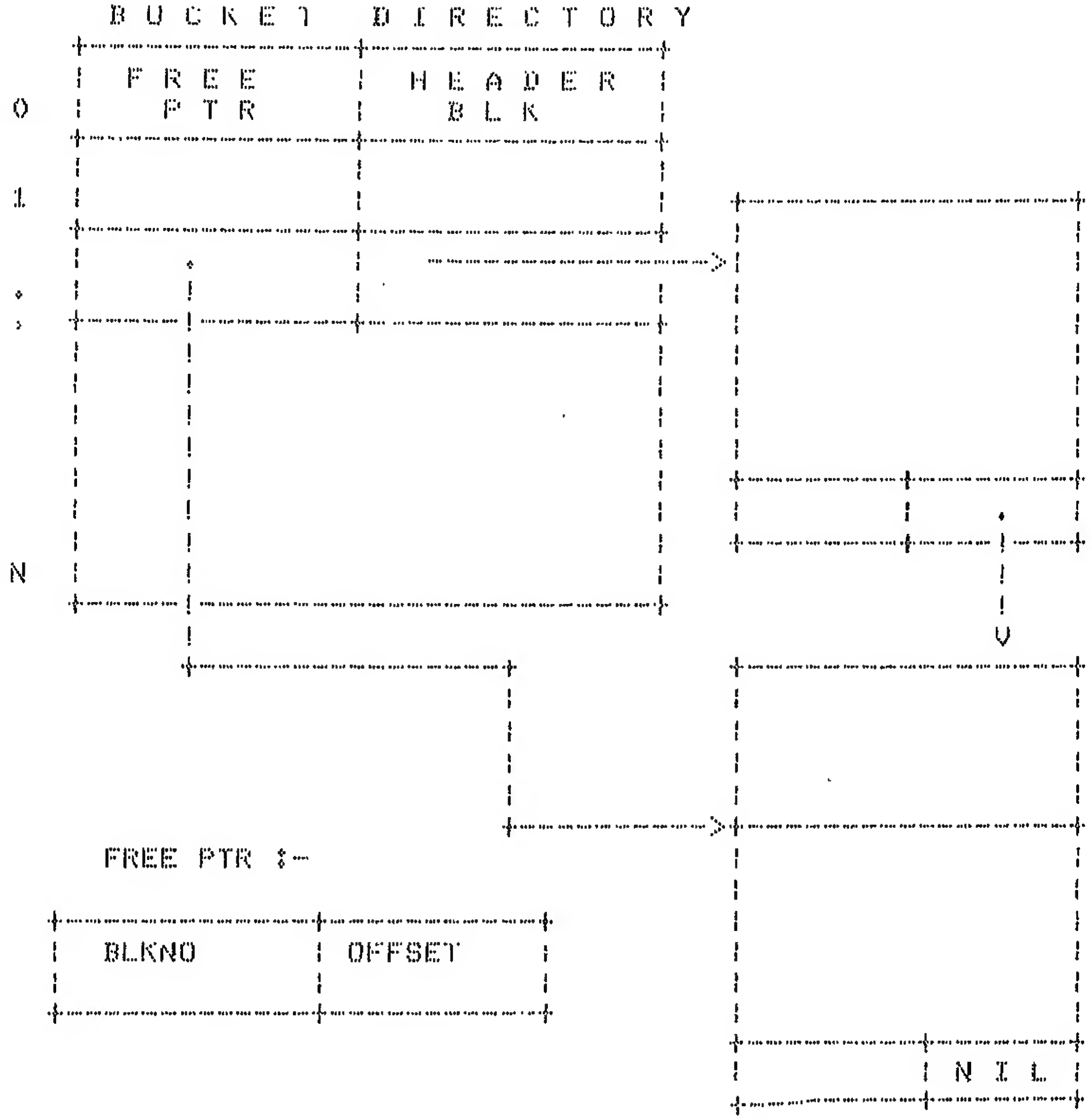

Fig. 3.1 : Hash File Structure

Each block has logically distinct data area and control information area. The data area stores the data records. The control information area stores the following control information :

(1) a field TOTALREC indicating total number of records in the block.

(2) a field NEXTBLK pointing to the next logical block.

(3) a BITMAP indicating whether the corresponding record in the block is deleted.

Fig. 3.2 describes the structure of each block in the buckets.

The first block is different from the rest of the blocks and is used to store control information for the file and the bucket directory.

Fig. 3.3 describes the first block which is different from the data blocks used by buckets.

```
+---------------------------------------+
|                                       |
|                                       |
|    D                                  |
|                                       |
|    A                                  |
|                                       |
|    T                                  |
|                   A                   |
|    A                                  |
|                   R                   |
|                                       |
|                   E                   |
|                                       |
|                   A                   |
+-------------------+-------------------+
| B I T M A P       |     N E X T       |
|                   |       B L K       |
+-------------------+-------------------+
```

Fig. 3.2 : A data block in hash file

C O N T R O L   B L O C K

H A S H (4 BYTES)
R E C S I Z E(2 BYTES)
NO. OF RECORDS/BLOCK(2BYTES)
NO. OF ATTRIBUTES(2 BYTES)
SIZE OF EACH ATTRIBUTE
KEY ATTRIBUTE(S)
B U C K E T D I R E C T O R Y (4 SECTORS)

Fig. 3.3 : Control block in hash file

It may be noted here that the real available space for the data in the data blocks is not 1K bytes as about 34 bytes are reserved for the two fields in the block shown in fig. 3.2.

The usual standard algorithms for look up, insertion, deletion, update of a record have been adopted. The set of routines for manipulation will be discussed in next chapter.

3.3 ISAM FILE STRUCTURE

The Indexed Sequential file attempts to overcome the problems encountered in ordinary sequential file but preserves all its benefits. It gives a better random access than the sequential file. In this the file is sorted in ascending order on its key value. An index is built on the value of key fields. Instead of searching the whole data area first the index area is scanned to get the zone of availability of the record in the file. The presence of an overflow pointer makes insertion easier.

In this the following are the design considerations :

(1) Primary data area

(2) Overflow data area structure

(3) Indexing level

The overall file structure consists of an index, a primary data area and an overflow data area as shown in fig. 3.4.

Blocks in primary data area have the following structure :

(1) data records with overflow pointers

(2) a free pointer to free area in the block

(3) a pointer to next logical block

(4) a BITMAP indicating whether the corresponding record is deleted.

The structure of a primary data area block is shown in fig. 3.5.

Blocks in overflow data area have almost the same structure except that instead of an overflow pointer we have a next record pointer as depicted in fig. 3.6.

Block level indexing has been adopted i.e. the index would have reference to the first record in the primary data block. Only one level of indexing would be enough because of maximum file size limitation in a diskette.

The index block structure is shown in Fig. 3.7. The first block in the file stores other control information like record size, total number of records, number of records per block, number of attributes and their respective sizes in bytes, key attributes in addition to the index entries as shown in fig.3.8.

A record pointer in this strucures has two parts (i) a block no. (ii) offset in the block.

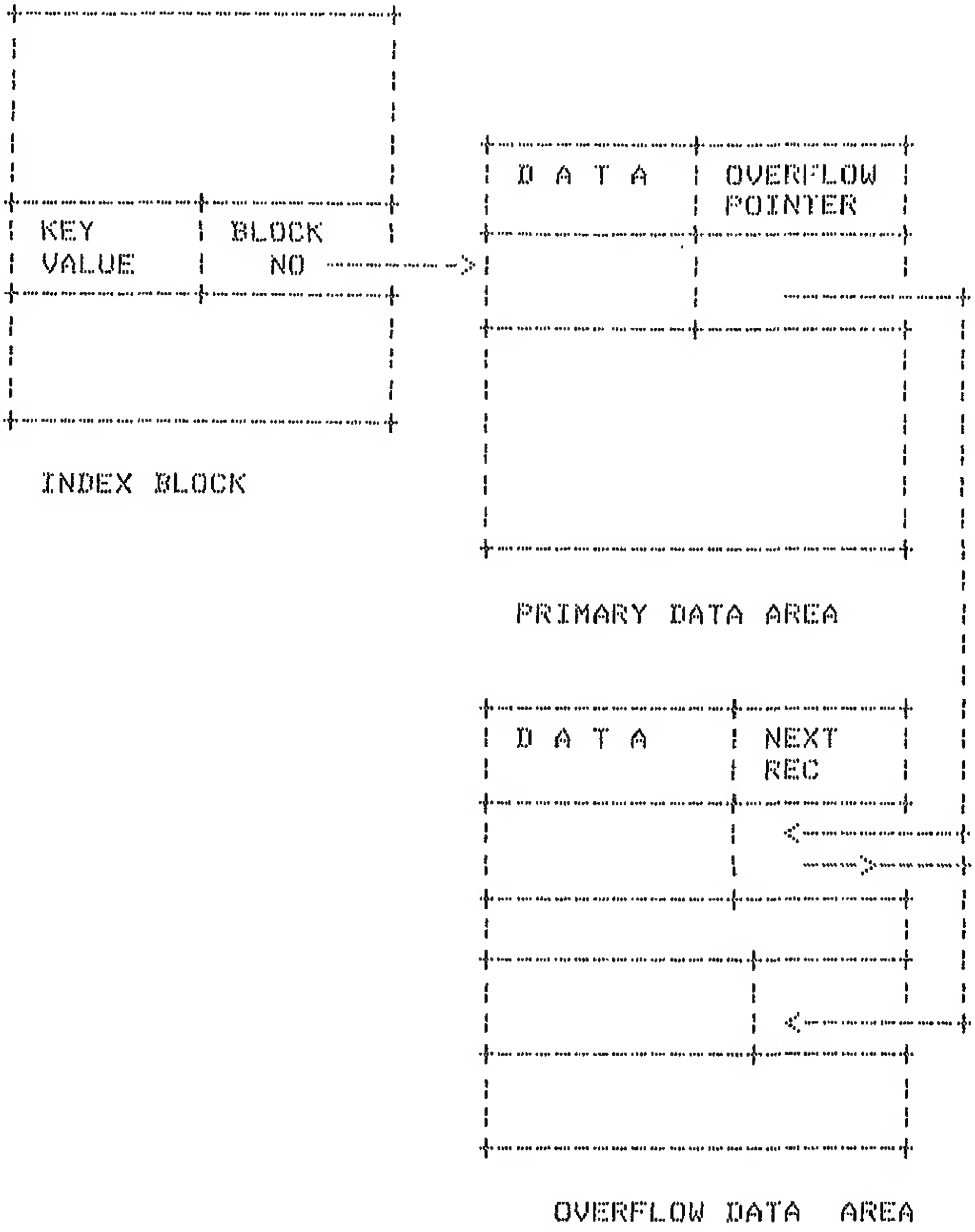

Fig. 3.4 : ISAM file structure

```
+--------------------------------+---------------+
|  D A T A                       |OVERFLOW       |
|          R E C O R D           |   PTR         |
+--------------------------------+---------------+
|                                                |
|                                                |
|                                                |
|                                                |
|                                                |
|                                                |
|                                                |
|                                                |
|                                                |
|                                                |
|                                                |
|                                                |
|                                                |
+------------------------------------------------+
|   B I T M A P   0=> VALID                      |
|                 1=> DELETED                    |
+------------------------+-----------------------+
|  F R E E               |   N E X T             |
|    P T R               |     B L K             |
+------------------------+-----------------------+
```

```
OVERFLOW PTR :-

+------------------------+-----------------------+
|                        |                       |
|         BLKNO          |        OFFSET         |
|                        |                       |
+------------------------+-----------------------+
```

Fig. 3.5 : A primary data block

```
+------------------------------+-----------+
|  D A T A                     |   NEXT    |
|         R E C O R D          |   RECORD  |
+------------------------------+-----------+
|                                          |
|                                          |
|                                          |
|                                          |
|                                          |
|                                          |
|                                          |
|                                          |
|                                          |
|                                          |
|                                          |
|                                          |
+------------------------------------------+
|    B I T M A P   0=> VALID               |
|                  1=> DELETED             |
+--------------------+---------------------+
|  F R E E           |   N E X T           |
|   P T R            |    B L K            |
+--------------------+---------------------+
```

NEXT RECORD :-

```
+--------------------+---------------------+
|                    |                     |
|       BLKNO        |       OFFSET        |
|                    |                     |
+--------------------+---------------------+
```

Fig. 3.6 : An overflow block

C O N T R O L I N F O R M A T I O N	
KEY VALUE	BLKNO
⋮	⋮

Fig. 3.7 : Control block in ISAM

C O N T R O L   I N F O R M A T I O N

I S A M (4 BYTES)
R E C S I Z E(2 BYTES)
NO. OF RECORDS/BLOCK(2BYTES)
NO. OF ATTRIBUTES(2 BYTES)
SIZE OF EACH ATTRIBUTE
KEY ATTRIBUTE(S)
I N D E X E N T R I E S (7 SECTORS)

Fig. 3.8 : Control information

The same procedure as described in previous section has been adopted to get an integer value from the key fields. The overflow blocks are also in the same file. No particular order is mainted in the overflow area. The records are not spanned over two different blocks.

A certain method is adopted to allocate an overflow block for the file as described below :

Every fifth block in the FCB is left as overflow blocks.If they are full then a new block is acquired at the current end of the file. This physical placement of overflow blocks would minimize the delay and improve the system response.

Standard algorithms for retrival, insertion, deletion, updating of a record have been adopted. The details of the routines for this are discussed in the next chapter.

3.4 ACCESS METHODS INTERFACE

As a design criterion of relational data base one of the major issues is to provide the user an independent view of the data free from the underlying file strucure. The access methods interface(AMI) is the interface between the user and the file structures providing the structural transperancy. This is a set of standard functions provided to the user to get access to the data stored in the data

base. There are ten basic function in the AMI of the current implementation. They are described in the following lines.

(1) OPENR(CHNO,FNAME)

This function opens the relation name specified in FNAME and assigns a channel number to it which is used in other routines of AMI. For any legal operation on the data base the relation has to be opened. This sets up all the control strucures for the corresponding file.

(2) GETNEXTRECORD(CHNO,ADRS)

This function reads the next tuple in the relation with channel no. CHNO and puts the record starting from ADRS.

(3) PUTNEXTRECORD(CHNO,ADRS)

This function writes at the next position in the relation with channel no. CHNO taking the record from location starting with ADRS.

(4) GETRECORD(CHNO,ADRS,KEYVAL)

This function retrieves the record with key value KEYVAL in the data base and if found puts it at the place pointed to by ADRS.

(5) INSERTRECORD(CHNO,ADRS)

This inserts a tuple to the relation with channel no. CHNO taking the tuple pointed by ADRS. It calculates the key value from the record itself

(6) DELETERECORD(CHNO,KEYVAL)

The record with key value KEYVAL is deleted from the relation with channel no. CHNO by setting the corresponding bit in the BITMAP.

(7) UPDATERECORD(CHNO,ADRS)

This function updates the tuple with key value as in the record pointed by ADRS and only updates the non key fields of the record.

(8) MODIFYRECORD(CHNO,ADRS) This is same as update function accept that it modifies the key fields too.

(9) CLOSER(CHNO) This function closes the relation with channel no CHNO for any further operation on it.

(10) CREATER(FNAME,FLAG,ATRCOUNT,ADRS1,ADRS2) This function creates a new relation with name as in FNAME and the file structure as specified by FLAG. If flag=0 then hashed structure is assumed, if flag=1 then ISAM file structure is assumed,if flag=2 then one of the two structures is chosen. The information about the no. of attributes is taken from ADRS1 and about key attributes from ADRS2. ATRCOUNT stands for total number of attributes in the relation.

CHAPTER 4

BASIC FUNCTIONAL LEVELS

4.1 FOUR BASIC LEVELS

In the implementation of the file structures discussed in previous chapter four distinct functional levels have been identified. In this chapter the four different levels would be discussed.

4.2 SECTOR LEVEL

This is the foundation level for the other functional levels that are built on top of it. In this level the set of routines directly interact with the BDOS(Basic Disk Operating System) module. As mentioned earlier BDOS provides many basic file handling routines which were required for building up the rest of the levels. Essentially this levels serves this purpose. This set of routines provide basic diskette management routines. These routines are not available to the users as these functions would not carry much sense for him in terms of his records. The current implementation on CDOS does not support random file handling facility directly. A pseudo-random facility has been provided on basic sequential files. In these

routines the unit of transfer is a sector(128 bytes).

Table 4.2.1 lists out basic routines in this level.

Table 4.2.1

NAME	FUNCTION
(i) CPMOPEN	Opens the file for any meaningful operation .
(ii) CPMMAKE	Creates a new file.
(iii) CPMSEAF	Searches for a file in directory.
(iv) CPMSETDMA	Sets the DMA address to a a specified value.
(v) CPMREADSEQ	Reads the next seq.sector at current DMA address.
(vi) CPMWRITESEQ	Writes the next seq. sector from current DMA address.
(vii) READRAND	Reads the given logical sector at specified adrs.
(viii) WRITERAND	Writes the logical sector from specified adrs.
(iX) CPMDELETE	Deletes a file from dir.
(X) CPMCLOSE	Closes the file from further operation on it.

4.3 BLOCK LEVEL

For any meaningful operation on a data base a sector wise transfer would cost a lot in terms of the secondary storage accesses. In order to minimize the secondary storage accesses a level with a higher unit is called for. This level has this purpose to serve. A block(1K bytes) has been chosen as the unit of transfer for this set of procedures as the file is stored in terms of blocks and the size is reasonable enough to reduce the secondary storage accesses. It provides almost all routines as in the previous level. Table 4.3.1 summarizes the set of routines provided by this level. This level is also transparent to the outside users.

Table 4.3.1

NAME	FUNCTION
(i) BOPEN	Block level open procedure
(ii) BREADSEQ	Reads the next seq. block in FCB into buffer.
(iii) BREADRAND	Reads the specified block (logical) in FCB into buf.
(iv) BWRITESEQ	Writes into next seq. block in FCB from buffer.

(v) BWRITERAND — Writes into the specified block in FCB from buffer.

(vi) BINSERT — Inserts a block at given place in FCB.

(vii) BDELETE — Deletes the specified block from FCB.

(viii) BCLOSE — Block level vlose routine.

4.4 RECORD LEVEL

This is the logical record level. A record is the logical unit of access for the user. The does not bother about the system dependent features like sector size or block size. His view of the data base is in terms of logical records. This level provides that interface to the data base. This set contains all basic routines to manipulate a data base like fetch, insert, delete, update, modify. The manipulation procedures being structure dependent this set contains two sets of procedures one for each file organization described in previous chapter. Table 4.4.1 describes the procedures and the associted function. For this level the unit of transaction as stated earlier is a record. This level makes extensive use of the previous two levels. An user with the knoweldge of the underlyingfile structures can use these routines.

Table 4.4.1

NAME	FUNCTION
(i) -----OPEN	Opens the file and sets up related data structures.
(ii) -----READSEQ	Reads the next logical rec.
(iii) -----WRITESEQ	Writes into the next logical record
(iv) -----FETCH	Fetches the record with specified key value.
(v) -----INSERT	Inserts the given record at proper place.
(vi) -----DELETE	Deletes the record with specified key value.
(vii) -----UPDATE	Updates the non-key fields of the specified record.
(viii) -----MODIFY	Modifies the whole record.
(ix) -----CLOSE	Closes the file and the associted data structure.

----- could be either HASH or ISAM.

4.5 RELATIONAL LEVEL

This is the top most level and uses the access method interface. The way it differs from the AMI is in terms of its ability to handle secondary index access. In this level the

unit of access is a logical record. However, this level has not been implemented and the details of the routines are to be decided when files based on secondary indices are provided.

CHAPTER 5

BUFFER MANAGEMENT

5.1 NEED FOR BUFFERS

A microcomputer has typically, 64K bytes of main memory. As stated earlier, the unit of transfer between diskette and the main memory is a sector of 128 bytes. An access to the diskette is the costliest operation time wise. To overcome these limitations a buffer scheme is called for. The scheme would load a block of data at a time rather than a sector so that the seek is minimized. As the main memory cannot provide an unlimited number of buffers, a fixed number of buffers are kept in a central pool and shared by all files in operation.

In the current implementation there is a pool of 32 1K byte buffers. The size of the buffer is chosen as 1K bytes to match the files in CP/M that are stored in terms of blocks. A buffer allocation algorithm must decide on a strategy for allocation of buffers to the files. The data structures used for this and the buffer allocation algorithm is given below.

5.2 DATA STRUCTURES

In order to utilize the buffers efficiently a good book-keeping of the status of each buffer over the query operation is required. For a typical query maximum of 16 differrent relations to be involved has been assumed. To maintain the status of each buffer different tables have been constructed. This section describes the tables in detail.

There are mainly three tables in operation namely (i) Table of Channels (TOC) (ii) Table of Files (TOF) (iii) Table of Buffers (TOB)

(i) Table Of Channels (TOC)

The structure of the table is shown in Fig. 5.1. Any file at the time of getting opened seeks for an entry in this table. This is indexed by CHNO which is the channel number assigned to the file succesfully opened. A channel number uniquely decides the file but not vice-versa. This table keeps ionformation about the current logical disk block in use and the offset in the block through two fields called 'CURDISKBLK' and 'OFFSET' respectively. FNO and BNO are two more fields in the table and are index numbers into the corresponding entry in the other two tables to be discussed. The last field in the table specifies whether the channel number is active or closed. The table has a

size of 16 entries. A function ALLOCCHANNEL allocates entries from this table at the time of file opening.

FNO	CURDISKBLK	OFFSET	BNO	CLOSED

Fig. 5.1 : Table Of Channels

(ii) Table Of Files (TOF)

Once a file is opened the relevent information about it is stored in this table as shown in Fig. 5.2. This table is indexed by FNO which appears as fields in other two tables. The FCB(File Control Block) of the file is stored in this table in the FCB field. CHNO is the channel number that has been assigned to this file is the second field. The number of buffers the relation is in possession from the central pool is indicated by BUFCOUNT field. This is used in the buffer allocation algorithm. The FTYPE field stores information regarding the type of file whether hashed or ISAM. As before the last field is the indicatdr for the

closed relations. There are 16 entries in this table. A function ALLOCFNO allocates entries from this table at the time of opening the relation.

FCB	CHNO	BUFCOUNT	FTYPE	CLOSED

Fig. 5.2 : Table Of Files

(iii) Table Of Buffers (TOB)

The structure of this table is shown in Fig. 5.3. This table keeps track of all the buffers. There are 32 entries in this table corresponding to each buffer of the buffer pool. BUFNO is the field indicating which buffer of the pool is allocated to this entry. This table is indexed by BNO which has already appeared in the other two tables. This table stores the corresponding CHNO and FNO of the relation in CHNO and FNO fields. DISKBNO indicates the logical block number in the FCB. The UPDATED flag is set if the buffer has been written into after it was last read from the diskette. The last field in the table indicates the

buffers that are free and available for use.

A function ALLOCBUF is availble which allocates buffers to a channel number as and when it asks for one, according to the buffer allocation algorithm to be discussed in next section.

BUFNO	FNO	DISKBNO	UPDATED	CHNO	FINISHED

Fig. 5.3 : Table Of Buffers

The three tables are maintained by all the top three levels of the last chapter. The sector level functions that do not need buffers ignore these tables.

5.3 BUFFER ALLOCATION ALGORITHM

As and when a relation requires a buffer it asks for one from the central pool throgh the ALLOCBUF function. Depending on availability of the buffers and the constraints put buffers are allotted from the pool. The strategy in case of non-availability of buffers is to penalize the file

with maximum number of buffers by removing one buffer from it. A minimum of two buffers are allocated to any opened file with out any constraint. Beyond two is decided by the algorithm given below.

INPUT: A valid channel no.

OUTPUT: An index to the pool of buffers or zero indicating that a buffer cannot be allotted to it. Hence it has to use its own buffer acquired before.

1.If TOF.BUFCOUNT>=2 then GOTO step 3 else CONTINUE. Either the relation is going to be opened (no buffers with it) or it has one buffer and now requires another one. In either case it has to be allotted one buffer as the CHANNEL NO. being valid implies that there are still less than 16 relations in operation.
Search for a buffer index that does not occur in TOB.BUFNO. IF found then allocate it to the channel and STOP ELSE GOTO step 2.

2. There is at least one relation with more than two buffers with it. Search for that entry in the TOF which has maximum number of buffers with the help of BUFCOUNT. It is the one to be penalized. Get the corresponding channel no of the entry found above. In TOB for the corresponding channel no. find the buffer that would involve least work i.e: get a buffer which has not been updated in that

channel no., if not found then take the first entry corresponding to the channel no. write it back into the disk and assign this buffer to the channel needing one and STOP. If a buffer without update is available then allocate the corresponding BUFNO and STOP.

3. If only free buffers available then allocate else return 0. Search for an index that does not occur in TOB.BUFNO. If found allocate the same and STOP
else return a zero indicating that a buffer is not available and STOP.

CHAPTER 6

PERFORMANCE ANALYSIS

6.1 PERFORMANCE MEASURES

In this chapter,performance analysis of the two file structures that have been designed is carried out. To compare the two file structures the following parameters have been chosen :

(i) Average Effective record size :R

(ii) Time to fetch a record given the key value :Tf

(iii) Time to get the next record within the file : Tn

(iv) Time to insert a record : Ti

(v) Time to update a record : Tu

Deleting a record is assumed to be just updating(setting) the corresponding delete bit in BITMAP,hence not considered as a measure.

The assumptions made and the symbols used in the derivation are given below :

(1) Record size=R bytes

(2) Block size= B bytes

(3) Unspanned records i.e. records are not split over two blocks

(4) Total no. of records in the file= n records

(5) Seek time= s, i.e., time required to position the head in right track

(6) Rotational latency= r i.e. time to locate the sector

(7) A block pointer size= P bytes

(8) A record pointer P'=P+Offset size

(9) Instantaneous transfer time =t

(10) Block transfer time= Btt= B/t

(11) Bulk transfer rate=t' (takes into account sector gap blocking,seek)

(12) The time to open a file,computation time to maintain the tables, and setting up of the control data structures are negligible to be ignored.

(13) Total no. of attributes in a record= a

(14) Average size of each attriburte= V bytes

(15) The index or the bucket directory is located in the main memory

(16) Bucket directory size (in hashed files)= d

(17) Computation time= c

(18) Expected Overflow chain length= Lc

(19) Probability of the record being in current block= Pc

6.2 AVERAGE EFFECTIVE RECORD SIZE

(a) HASHED

R=R1+R2+R3

where R1=Actual size of attributes= aV bytes

R2= Control information per record = (B/n)

R3= Wasted space overhead per record

= (B-Floor(B/aV)*aV)/(Floor(B/aV))

(b) ISAM

R= R1+R2+R3+R4

where R1= actual size of record= aV bytes

R2= Record pointer= P'

R3= Space occupied by index blocks per record=(B/n)

R4= Wastage over head per record

=(B-Floor(B/(aV+P'))*(aV+P'))/(Floor(B/(aV+P')))

6.3 TIME TO FETCH A RECORD

(a) HASHED

Tf= T1+T2+T3

where T1= Computation time to get the pointer to block=c

T2= Reading Block(s) in the bucket

= Time to read average no. of blocks in a bucket/2

= (n/d)*(s+r+btt)/(2*(Floor(B/aV)))

T3= Search time in the blocks of the bucket

= O(n'Logn')

(b) ISAM

Tf= T1+T2+T3

where T1= Time to get the block no. from index= c

T2= Time to read the block(s) in the Chain

= Expected Overflow chain length*time to read a block

= Lc*(s+r+btt)

T3= Time to search in blocks

= O(n'Logn')

6.4 TIME TO GET NEXT RECORD

(a) HASHED

Tn= Tf if the key value of the next record is known

else

= reading next record in buffer

= P*c+(1-p)*(s+r+btt)

(b) ISAM

In this case it would depend on where the last record was. There are four possibilities :

(i) The last record and next record are in primary area.

Tn= p*c+(1-p)*(s+r+btt)

(ii) The last record was in primary and next record is in overflow area

Tn= c+(s+r+btt)

(iii) The last record was in overflow area and next record is in primary area.

Tn= c+(s+r+btt)

(iv) The last record and the next record are in overflow area

Tn= p*c+(1-p)*(s+r+btt)

6.5 TIME TO INSERT A RECORD

(a) HASHED

Ti= T1+T2+T3

where T1= locate the free pointer from bucket directory= C1

T2= read the block into buffer=(s+r+btt)

T3= Write into buffer

= C2

(b) ISAM

Ti= Time To locate the place+write into buffer = Tf+c

6.6 TIME TO UPDATE A RECORD

In case only the non-key fields are updated :

(a) HASHED

Tu=Locate the record+ write into buffer

=Tf+c

(b) ISAM

Tu=Locate the record+write into the buffer

= Tf+c

In case the key fields being updated, the time in both cases would be given by

T =Ti+Tu

In the above derivations we have assumed a very simple model. For a floppy based systems many other parameters come into picture like Head loading time, Head settling

time, Motor start time etc. in hardware parameters and Operating system characteristics in handling the file and pattern of access by the user program. But incorporating all these would make the model very complicated. An interesting experimental model taking into account all such parameters have been described in [PECHU83].

CHAPTER 7

CONCLUSION

7.1 SUMMARY

In this thesis the major goal has been to implement an efficient storage structure for RDB/M -a relational data base for microcomputers with CP/M as the operating system.

Two types of file organizations have been provided viz. (i) HASHED (ii) ISAM. The structures of these two files for the particular implementation has been discussed. In implementing the file system four basic functional levels have been identified. The basic two levels are common to any file structure that in future would be included.

To upgrade system performance a buffer management scheme has been adopted. The details of the data structures used and the buffer allocation algorithm is discussed.

A performance analysis of both the file structures has been carried out analytically. Few performance measures have been chosen and the same for both has been computed.

The implementation has been carried out on CROMEMCO SYSTEM THREE,on its CDOS operating system. Information regarding CDOS-CP/M compatibility is available in [CDOSUM]. The language used for the implementation is a special version of PASCAL called PASCAL/MT[MTPAS].

7.2 SCOPE FOR FUTURE WORK

This has been an initial attempt to implement a relational data base on microcomputers. As a first step towards this, four important features of a relational data base have been chosen. There are lot more things to be done before it could become commercially usable.

As an initial extension modifications in the present implementation be made to incorporate WINCHESTER drives as secondary storage device.

In the storage structure, protection of the data base from unauthorized use is to be provided. A standard model to extend the existing strucure for concurrency control can be thought of. Facilities for selective access control and Providing User views and snap shots are going to be some interesting features for implementation. Some other typical secondary key based file structures should be provided.

Only an analytical performance analysis has been carried out. An experimental model to measure these for a particular system as suggested in[PECH83] could be carried out to measure different performance measures.

REFERENCES

CDOSUM Cromemco CDOS users manual ,Cromemco, Inc. 1980

CHAMB76 Chamberlin, D. D.: Relational data base management systems. ACM comput. Survey 8,1(Mar. 76),43-67

CODD70 Codd, E.F.: A relational model of data for large shared data banks. CACM 13,6(Jun 70),377-397

CODD79 Codd, E.F.: Extending the data base relational model to capture more meaning. ACM TODS 4,4(Dec. 79),

CPMIG CP/M 2.0 Interface Guide. Digital Research 1979.

CPMUG CP/M 2.2 Users Guide. Digital Research 1979.

CPMAG CP/M 2.2 Alteration Guide. Digital Research 1979

CPMF An introduction to CP/M features and facilities. Digital Reseach 1979

FRSB76 Fry, James P. and Sibley, Edger H. : Evolution of DBMS. ACM comput. Survey 8,1(Mar. 76),7-42.

KIM79 Kim Won : Relational Data Base Systems. ACM comput. Survey 11,3(Sept. 79),185-213

KNU3 Knuth, Donald E.: The art of computer programming Vol. 3 ,Sorting and Searching .

MILLER Miller, Alan R. :Mastering CP/M . SYBEX 1981

MTPAS PASCAL/MT 3.0 Guide . MT MicroSYSTEMS

PECH83 Pechura, Michael A. : Comparing two microcomputer OSs:CP/M and HDOS ,CACM 26,3(Mar. 83),188-195.

PECHU83 Pechura Michael A. and Schoeffler, J. d.; Estimating File access time of floppy disks . CACM 26,10(Oct.83),754-763.

STONE76 Stonebraker, Michael.,Wong Eugene.,Kreps Peter and Held,Gerald.; The design and implementation of INGRES. ACM TODS 1,3(Sep. 76),189-222

ULLMAN2 Ullman,J.D.;Principles of data base systems. Second Edn.Galgotia Pub.,1983.

WIED2 Wiederhold, Gio ; Data base systems. Second Edn., McGraw Hill Book Co.,1984.